US009258390B2

(12) **United States Patent**
Pope et al.

(10) **Patent No.: US 9,258,390 B2**
(45) **Date of Patent: Feb. 9, 2016**

(54) **REDUCING NETWORK LATENCY**

(75) Inventors: **Steven L. Pope**, Costa Mesa, CA (US); **David J. Riddoch**, Huntingdon (GB); **Kieran Mansley**, Girton (GB)

(73) Assignee: **SOLARFLARE COMMUNICATIONS, INC.**, Irvine, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 981 days.

(21) Appl. No.: **13/283,420**

(22) Filed: **Oct. 27, 2011**

(65) **Prior Publication Data**

US 2013/0031268 A1 Jan. 31, 2013

Related U.S. Application Data

(60) Provisional application No. 61/513,108, filed on Jul. 29, 2011.

(51) **Int. Cl.**
G06F 15/16 (2006.01)
H04L 29/06 (2006.01)

(52) **U.S. Cl.**
CPC ***H04L 69/22*** (2013.01); *H04L 69/166* (2013.01)

(58) **Field of Classification Search**
CPC G06F 3/1243; H04L 15/5825; H04L 29/08081
USPC 709/237
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,272,599 A 12/1993 Koenen
5,325,532 A 6/1994 Crosswy et al.
5,946,189 A 8/1999 Koenen et al.
6,098,112 A 8/2000 Ishijima et al.
6,160,554 A 12/2000 Krause
6,304,945 B1 10/2001 Koenen
6,349,035 B1 2/2002 Koenen
6,427,173 B1 7/2002 Boucher et al.
6,438,130 B1 8/2002 Kagan et al.

(Continued)

FOREIGN PATENT DOCUMENTS

EP 620521 A2 10/1994
EP 2463782 A2 6/2012

(Continued)

OTHER PUBLICATIONS

EP 12185546.4—Extended European Search Report dated Jul. 13, 2013, 6 pages.

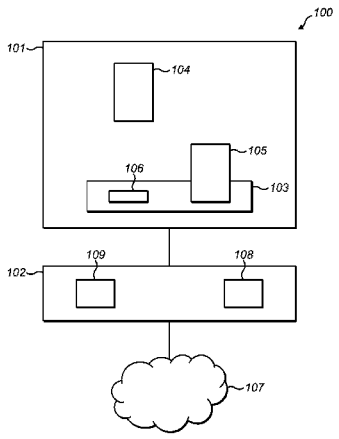
(Continued)

Primary Examiner — Arvin Eskandarnia
(74) *Attorney, Agent, or Firm* — Haynes Beffel & Wolfeld LLP; Warren S. Wolfeld

(57) **ABSTRACT**

A method of transmitting data for use at a data processing system and network interface device, the data processing system being coupled to a network by the network interface device, the method comprising: forming a message template in accordance with a predetermined set of network protocols, the message template including at least in part one or more protocol headers; forming an application layer message in one or more parts; updating the message template with the parts of the application layer message; processing the message template in accordance with the predetermined set of network protocols so as to complete the protocol headers; and causing the network interface device to transmit the completed message over the network.

36 Claims, 2 Drawing Sheets



(56) **References Cited**

U.S. PATENT DOCUMENTS

6,502,203 B2 12/2002 Barron et al.
6,530,007 B2 3/2003 Olarig et al.
6,591,302 B2 7/2003 Boucher et al.
6,667,918 B2 12/2003 Leader et al.
6,718,392 B1 4/2004 Krause
6,728,743 B2 4/2004 Shachar
6,735,642 B2 5/2004 Kagan et al.
6,768,996 B1 7/2004 Steffens et al.
6,904,534 B2 6/2005 Koenen
6,950,961 B2 9/2005 Krause et al.
6,965,941 B2 11/2005 Boucher et al.
6,978,331 B1 12/2005 Kagan et al.
7,089,326 B2 8/2006 Boucher et al.
7,093,158 B2 8/2006 Barron et al.
7,099,275 B2 8/2006 Sarkinen et al.
7,103,626 B1 9/2006 Recio et al.
7,103,744 B2 9/2006 Garcia et al.
7,136,397 B2 11/2006 Sharma
7,143,412 B2 11/2006 Koenen
7,149,227 B2 12/2006 Stoler et al.
7,151,744 B2 12/2006 Sarkinen et al.
7,216,225 B2 5/2007 Haviv et al.
7,240,350 B1 7/2007 Eberhard et al.
7,245,627 B2 7/2007 Goldenberg et al.
7,254,237 B1 8/2007 Jacobson et al.
7,285,996 B2 10/2007 Fiedler
7,316,017 B1 1/2008 Jacobson et al.
7,346,702 B2 3/2008 Haviv
7,386,619 B1 6/2008 Jacobson et al.
7,403,535 B2 7/2008 Modi et al.
7,404,190 B2 7/2008 Krause et al.
7,502,826 B2 3/2009 Barron et al.
7,502,870 B1 3/2009 Chu
7,509,355 B2 3/2009 Hanes et al.
7,518,164 B2 4/2009 Smelloy et al.
7,551,614 B2 6/2009 Teisberg et al.
7,554,993 B2 6/2009 Modi et al.
7,573,967 B2 8/2009 Fiedler
7,580,415 B2 8/2009 Hudson et al.
7,580,495 B2 8/2009 Fiedler
7,617,376 B2 11/2009 Chadalapaka et al.
7,631,106 B2 12/2009 Goldenberg et al.
7,636,703 B2 12/2009 Taylor
7,650,386 B2 1/2010 McMahan et al.
7,653,754 B2 1/2010 Kagan et al.
7,688,853 B2 3/2010 Santiago et al.
7,702,629 B2 4/2010 Cytron et al.
7,725,556 B1 5/2010 Schlansker et al.
7,757,232 B2 7/2010 Hilland et al.
7,801,027 B2 9/2010 Kagan et al.
7,802,071 B2 9/2010 Oved
7,813,460 B2 10/2010 Fiedler
7,827,442 B2 11/2010 Sharma et al.
7,835,375 B2 11/2010 Sarkinen et al.
7,835,380 B1 11/2010 Aloni et al.
7,848,322 B2 12/2010 Oved
7,856,488 B2 12/2010 Cripe et al.
7,864,787 B2 1/2011 Oved
7,895,445 B1 2/2011 Albanese et al.
7,904,576 B2 3/2011 Krause et al.
7,921,178 B2 4/2011 Haviv
7,929,539 B2 4/2011 Kagan et al.
7,930,437 B2 4/2011 Kagan et al.
7,934,959 B2 5/2011 Rephaeli et al.
7,945,528 B2 5/2011 Cytron et al.
7,954,114 B2 5/2011 Chamberlain et al.
7,978,606 B2 7/2011 Buskirk et al.
8,000,336 B2 8/2011 Harel
8,156,101 B2 4/2012 Indeck et al.
8,286,193 B2 10/2012 Pope et al.
8,326,816 B2 12/2012 Colle et al.
2002/0059052 A1 5/2002 Bloch et al.
2002/0095519 A1 7/2002 Philbrick et al.
2002/0112139 A1 8/2002 Krause et al.
2002/0129293 A1 9/2002 Hutton et al.
2002/0140985 A1 10/2002 Hudson
2002/0156784 A1 10/2002 Hanes et al.
2002/0174240 A1* 11/2002 Nason H04L 29/06 709/230
2002/0198990 A1 12/2002 Bradfield et al.
2003/0007165 A1 1/2003 Hudson
2003/0033588 A1 2/2003 Alexander
2003/0058459 A1 3/2003 Wu et al.
2003/0063299 A1 4/2003 Cowan et al.
2003/0065856 A1 4/2003 Kagan et al.
2003/0081060 A1 5/2003 Zeng et al.
2003/0086300 A1 5/2003 Noyes et al.
2003/0172330 A1 9/2003 Barron et al.
2003/0191786 A1 10/2003 Matson et al.
2003/0202043 A1 10/2003 Zeng et al.
2003/0214677 A1 11/2003 Bhaskar et al.
2004/0015502 A1 1/2004 Alexander et al.
2004/0071250 A1 4/2004 Bunton et al.
2004/0141642 A1 7/2004 Zeng et al.
2004/0156393 A1 8/2004 Gupta et al.
2004/0190533 A1 9/2004 Modi et al.
2004/0190538 A1 9/2004 Bunton et al.
2004/0190557 A1 9/2004 Barron
2004/0193734 A1 9/2004 Barron et al.
2004/0193825 A1 9/2004 Garcia et al.
2004/0210670 A1 10/2004 Anerousis et al.
2004/0210754 A1 10/2004 Barron et al.
2004/0240435 A1 12/2004 Boucher et al.
2004/0249881 A1 12/2004 Jha et al.
2004/0249998 A1 12/2004 Rajagopalan et al.
2004/0252685 A1 12/2004 Kagan et al.
2005/0008223 A1 1/2005 Zeng et al.
2005/0018221 A1 1/2005 Zeng et al.
2005/0021874 A1 1/2005 Georgiou et al.
2005/0038918 A1 2/2005 Hilland et al.
2005/0038941 A1 2/2005 Chadalapaka et al.
2005/0039171 A1 2/2005 Avakian et al.
2005/0039172 A1 2/2005 Rees et al.
2005/0039187 A1 2/2005 Avakian et al.
2005/0066333 A1 3/2005 Krause et al.
2005/0172181 A1 8/2005 Huliehel
2005/0219278 A1 10/2005 Hudson
2005/0219314 A1 10/2005 Donovan et al.
2005/0231751 A1 10/2005 Wu et al.
2006/0026443 A1 2/2006 McMahan et al.
2006/0045098 A1 3/2006 Krause
2006/0126619 A1 6/2006 Teisberg et al.
2006/0165074 A1 7/2006 Modi et al.
2006/0193318 A1 8/2006 Narasimhan et al.
2006/0228637 A1 10/2006 Jackson et al.
2006/0248191 A1 11/2006 Hudson et al.
2007/0121596 A1* 5/2007 Kurapati H04L 29/06027 370/356
2007/0188351 A1 8/2007 Brown et al.
2007/0209069 A1* 9/2007 Saklikar H04L 63/1433 726/14
2007/0220183 A1 9/2007 Kagan et al.
2007/0237327 A1 10/2007 Taylor et al.
2007/0260602 A1 11/2007 Taylor
2007/0277036 A1 11/2007 Chamberlain et al.
2008/0008205 A1 1/2008 Jung et al.
2008/0024586 A1 1/2008 Barron
2008/0109526 A1 5/2008 Subramanian et al.
2008/0115216 A1 5/2008 Barron et al.
2008/0115217 A1 5/2008 Barron et al.
2008/0126509 A1 5/2008 Subramanian et al.
2008/0135774 A1 6/2008 Hugers
2008/0140574 A1 6/2008 Boucher et al.
2008/0147828 A1 6/2008 Enstone et al.
2008/0148400 A1 6/2008 Barron et al.
2008/0177890 A1 7/2008 Krause et al.
2008/0244060 A1 10/2008 Cripe et al.
2008/0301406 A1 12/2008 Jacobson et al.
2008/0304519 A1 12/2008 Koenen et al.
2009/0060197 A1 3/2009 Taylor et al.
2009/0165003 A1 6/2009 Jacobson et al.
2009/0182683 A1 7/2009 Taylor et al.
2009/0183057 A1 7/2009 Aizman
2009/0201926 A1 8/2009 Kagan et al.

(56) **References Cited**

U.S. PATENT DOCUMENTS

2009/0213856	A1	8/2009	Paatela et al.	
2009/0268612	A1	10/2009	Felderman et al.	
2009/0287628	A1	11/2009	Indeck et al.	
2009/0302923	A1	12/2009	Smeloy et al.	
2010/0088437	A1	4/2010	Zahavi	
2010/0138840	A1	6/2010	Kagan et al.	
2010/0169880	A1	7/2010	Haviv et al.	
2010/0188140	A1	7/2010	Smeloy	
2010/0189206	A1	7/2010	Kagan	
2010/0198850	A1	8/2010	Cytron et al.	
2010/0265849	A1	10/2010	Harel	
2010/0274876	A1	10/2010	Kagan et al.	
2011/0004457	A1	1/2011	Haviv et al.	
2011/0010557	A1	1/2011	Kagan et al.	
2011/0029669	A1	2/2011	Chuang et al.	
2011/0029847	A1	2/2011	Goldenberg et al.	
2011/0040701	A1	2/2011	Singla et al.	
2011/0044344	A1	2/2011	Hudson et al.	
2011/0058571	A1	3/2011	Bloch et al.	
2011/0083064	A1	4/2011	Kagan et al.	
2011/0096668	A1	4/2011	Bloch et al.	
2011/0113083	A1	5/2011	Shahar	
2011/0116512	A1	5/2011	Crupnicoff et al.	
2011/0119673	A1	5/2011	Bloch et al.	
2011/0173352	A1	7/2011	Sela et al.	
2011/0178917	A1	7/2011	Parsons et al.	
2011/0178918	A1	7/2011	Parsons et al.	
2011/0178919	A1*	7/2011	Parsons	G06Q 40/00 705/37
2011/0178957	A1	7/2011	Parsons et al.	
2011/0184844	A1	7/2011	Parsons et al.	
2012/0089496	A1	4/2012	Taylor et al.	
2012/0089497	A1	4/2012	Taylor et al.	
2012/0095893	A1	4/2012	Taylor et al.	
2012/0102245	A1	4/2012	Gole et al.	
2012/0246052	A1	9/2012	Taylor et al.	
2013/0007000	A1	1/2013	Indeck et al.	

FOREIGN PATENT DOCUMENTS

WO	00/10095	A1	2/2000
WO	0148972	A1	7/2001
WO	0235838	A1	5/2002
WO	2008127672	A2	10/2008
WO	2009134219	A1	11/2009
WO	2009136933	A1	11/2009
WO	2010020907	A2	2/2010
WO	2010087826	A1	8/2010
WO	2011043769	A1	4/2011
WO	2011053305	A1	5/2011
WO	2011053330	A1	5/2011

OTHER PUBLICATIONS

EP 13187725.0—1953—Extended European Search Report dated Feb. 19, 2014, (6 pages).
EP 13153148.5—1953—Extended European Search Report dated Feb. 19, 2014, 6 pages.
Gordon E. Moore; Electronics, vol. 38, No. 8, pp. 114-117, 1965, Apr. 19, 1965.
Jack B. Dennis and Earl C. Van Horn; Communications of the ACM, vol. 9, No. 3, pp. 143-155, 1966, Mar. 1966.
Marvin Zelkowitz; Communications of the ACM, vol. 14, No. 6, p. 417-418, 1971, Jun. 1971.
J. Carver Hill; Communications of the ACM, vol. 16, No. 6, p. 350-351, 1973, Jun. 1973.
F.F. Kuo; ACM Computer Communication Review, vol. 4 No. 1, 1974, Jan. 1974.
Vinton Cerf, Robert Kahn; IEEE Transactions on Communications, vol. COM-22, No. 5, pp. 637-648, 1974, May 1974.
V. Ceti, et al.; ACM Computer Communication Review, vol. 6 No. 1, p. 1-18, 1976, Jan. 1976.
Robert M. Metcalfe and David R. Boggs; Communications of the ACM, vol. 19, Issue 7, pp. 395-404, 1976, Jul. 1976.
P. Kermani and L. Kleinrock; Computer Networks, vol. 3, No. 4, pp. 267-286, 1979, Sep. 1979.
John M. McQuillan, et al.; Proceedings of the 6th Data Communications Symposium, p. 63, 1979, Nov. 1979.
Andrew D. Birrell, et al.; Communications of the ACM, vol. 25, Issue 4, pp. 260-274, 1982, Apr. 1982.
Ian M. Leslie, et al.; ACM Computer Communication Review, vol. 14, No. 2, pp. 2-9, 1984, Jun. 1984.
John Nagle; ACM Computer Communication Review, vol. 14, No. 4, p. 11-17, 1984, Oct. 1984.
Robert M. Brandriff, et al.; ACM Computer Communication Review, vol. 15, No. 4, 1985, Sep. 1985.
C. Kline; ACM Computer Communication Review, vol. 17, No. 5, 1987, Aug. 1987.
Christopher A. Kent, Jeffrey C. Mogul; ACM Computer Communication Review, vol. 17, No. 5, pp. 390-401, 1987, Oct. 1987.
Gary S. Delp, et al.; ACM Computer Communication Review, vol. 18, No. 4, p. 165-174, 1988, Aug. 1988.
David R. Boggs, et al.; ACM Computer Communication Review, vol. 18, No. 4, p. 222-234, 1988, Aug. 1988.
H. Kanakia and D. Cheriton; ACM Computer Communication Review, vol. 18, No. 4, p. 175-187, 1988, Aug. 1988.
V. Jacobson; ACM Computer Communication Review, vol. 18, No. 4, p. 314-329, 1988, Aug. 1988.
David D. Clark; ACM Computer Communication Review, vol. 18, No. 4, pp. 106-114, 1988, Aug. 1988.
Paul V. Mockapetris, Kevin J. Dunlap; ACM Computer Communication Review, vol. 18, No. 4, pp. 123-133, 1988, Aug. 1988.
Margaret L. Simmons and Harvey J. Wasserman; Proceedings of the 1988 ACM/IEEE conference on Supercomputing, p. 288-295, Orlando, Florida, Nov. 12, 1988.
David A. Borman; ACM Computer Communication Review, vol. 19, No. 2, p. 11-15, 1989, Apr. 1989.
R. Braden, et al.; ACM Computer Communication Review, vol. 19, No. 2, p. 86-94, 1989, Apr. 1989.
David D. Clark, et al.; IEEE Communications Magazine, vol. 27, No. 6, pp. 23-29, 1989, Jun. 1989.
David R. Cheriton; ACM Computer Communication Review, vol. 19, No. 4, p. 158-169, 1989, Sep. 1989.
Derek Robert McAuley; PhD Thesis, University of Cambridge, 1989, Sep. 1989.
Craig Partridge; ACM Computer Communication Review, vol. 20, No. 1, p. 44-53, 1990, Jan. 1990.
D. D. Clark and D. L. Tennenhouse; ACM Computer Communication Review, vol. 20, No. 4, pp. 200-208, 1990, Sep. 1990.
Eric C. Cooper, et al.; ACM Computer Communication Review, vol. 20, No. 4, p. 135-144, 1990, Sep. 1990.
Bruce S. Davie; ACM Computer Communication Review, vol. 21, No. 4, 1991, Sep. 1991.
C. Brendan S. Traw, et al.; ACM Computer Communication Review, vol. 21, No. 4, p. 317-325, 1991, Sep. 1991.
Ian Leslie and Derek R. McAuley; ACM Computer Communication Review, vol. 21, No. 4, p. 327, 1991, Sep. 1991.
Mark Hayter, Derek McAuley; ACM Operating Systems Review, vol. 25, Issue 4, p. 14-21, 1991, Oct. 1991.
Gregory G. Finn; ACM Computer Communication Review, vol. 21, No. 5, p. 18-29, 1991, Oct. 1991.
Greg Chesson; Proceedings of the Third International Conference on High Speed Networking, 1991, Nov. 1991.
Michael J. Dixon; University of Cambridge Computer Laboratory Technical Report No. 245, Jan. 1992.
Danny Cohen, Gregory Finn, Robert Felderman, Annette DeSchon; Made available by authors, Jan. 10, 1992.
Gene Tsudik; ACM Computer Communication Review, vol. 22, No. 5, pp. 29-38, 1992, Oct. 1992.
Peter Steenkiste; ACM Computer Communication Review, vol. 22, No. 4, 1992, Oct. 1992.
Paul E. McKenney and Ken F. Dove; ACM Computer Communication Review, vol. 22, No. 4, 1992, Oct. 1992.
Erich Ruetsche and Matthias Kaiserswerth; Proceedings of the IFIP TC6/WG6.4 Fourth International Conference on High Performance Networking IV, Dec. 14, 1992.

(56) **References Cited**

OTHER PUBLICATIONS

C. Traw and J. Smith; IEEE Journal on Selected Areas in Communications, pp. 240-253, 1993, Feb. 1993.
E. Ruetsche; ACM Computer Communication Review, vol. 23, No. 3, 1993, Jul. 1993.
Jonathan M. Smith and C. Brendan S. Traw; IEEE Network, vol. 7, Issue 4, pp. 44-52, 1993, Jul. 1993.
Jeffrey R. Michel; MSci Thesis, University of Virginia, 1993, Aug. 1993.
Mark David Hayter; PhD Thesis, University of Cambridge, 1993, Sep. 1993.
Jonathan Kay and Joseph Pasquale; ACM Computer Communication Review, vol. 23, No. 4, pp. 259-268, 1993, Oct. 1993.
W. E. Leland, et al.; ACM Computer Communication Review, vol. 23, No. 4, p. 85-95, 1993, Oct. 1993.
Regnier G., "Protocol Onload vs. Offload," 14th Symposium on High Performance Interconnects, Aug. 23, 2006, 1pp.
Montry G., OpenFabrics Alliance presentation slides, 14th Symposium on High Performance Interconnects, Aug. 23, 2006, 8pp.
Bilic Hrvoye, et al.; article in Proceedings of the 9th Symposium on High Performance Interconnects, "Deferred Segmentation for Wire-Speed Transmission of Large TCP Frames over Standard GbE Networks," Aug. 22, 2001, 5pp.
Bilic Hrvoye, et al.; presentation slides from 9th Symposium on High Performance Interconnects, "Deferred Segmentation for Wire-Speed Transmission of Large TCP Frames over Standard GbE Networks," Aug. 22, 2001, 9pp.
Bruce Lowekamp, et al.; ACM Computer Communication Review, vol. 31, No. 4, 2001, Oct. 2001.
Piyush Shivam, et al.; Proceedings of the 2001 ACM/IEEE conference on Supercomputing, pp. 57, Denver, Nov. 10, 2001.
Robert Ross, et al.; Proceedings of the 2001 ACM/IEEE conference on Supercomputing, pp. 11, Denver, Nov. 10, 2001.
E. Blanton and M. Allman; ACM Computer Communication Review, vol. 32, No. 1, 2002, Jan. 2002.
Murali Rangarajan, et al.; Technical Report DCR-TR-481, Computer Science Department, Rutgers University, 2002. , Mar. 2002.
Jon Crowcroft, Derek McAuley; ACM Computer Communication Review, vol. 32, No. 5, 2002, Nov. 2002.
Charles Kalmanek; ACM Computer Communication Review, vol. 32, No. 5, pp. 13-19, 2002, Nov. 2002.
Jonathan Smith; ACM Computer Communication Review, vol. 32, No. 5, pp. 29-37, 2002, Nov. 2002.
NR Adiga, et al.; Proceedings of the 2002 ACM/IEEE conference on Supercomputing, pp. 1-22, Baltimore, Nov. 16. 2002.
Steven J. Sistare, Christopher J. Jackson; Proceedings of the 2002 ACM/IEEE conference on Supercomputing, p. 1-15, Baltimore, Nov. 16, 2002.
R. Bush, D. Meyer; IETF Network Working Group, Request for Comments: 3439, Dec. 2002.
Pasi Sarolahti, et al.; ACM Computer Communication Review, vol. 33, No. 2, 2003, Apr. 2003.
Tom Kelly; ACM Computer Communication Review, vol. 33, No. 2, pp. 83-91, 2003, Apr. 2003.
Jeffrey C. Mogul; Proceedings of HotOS IX: The 9th Workshop on Hot Topics in Operating Systems, pp. 25-30, May 18, 2003.
Derek McAuley, Rolf Neugebauer; Proceedings of the ACM SIGCOMM 2003 Workshops, Aug. 2003.
Justin Hurwitz, Wu-chun Feng; Proceedings of the 11th Symposium on High Performance Interconnects, Aug. 20, 2003.
Vinay Aggarwal, et al.; ACM Computer Communication Review, vol. 33, No. 5, 2003, Oct. 2003.
Wu-chun Feng, et al.; Proceedings of the 2003 ACM/IEEE conference on Supercomputing, Phoenix, Arizona, Nov. 15, 2003.
Jiuxing Liu, et al.; Proceedings of the 2003 ACM/IEEE conference on Supercomputing, Phoenix, Arizona, Nov. 15, 2003.
Srihari Makineni and Ravi Iyer; Proceedings of the 10th International Symposium on High Performance Computer Architecture, pp. 152, Feb. 14, 2004.
Cheng Jin, et al.; Proceedings of IEEE Infocom 2004, pp. 1246-1259, Mar. 7, 2004.
Andy Currid; ACM Queue, vol. 2, No. 3, 2004, May 1, 2004.
Greg Regnier, et al.; Computer, IEEE Computer Society, vol. 37, No. 11, pp. 48-58, 2004, Nov. 2004.
Gregory L. Chesson; United States District Court, Northern District California, San Francisco Division, Feb. 4, 2005.
Edward D. Lazowska, David A. Patterson; ACM Computer Communication Review, vol. 35, No. 2, 2005, Jul. 2005.
W. Feng, et al.; Proceedings of the 13th Symposium on High Performance Interconnects, Aug. 17, 2005.
B. Leslie, et al.; J. Comput. Sci. & Technol., vol. 20, Sep 2005, Sep. 2005.
P. Balaji, et al.; Proceedings of the IEEE International Conference on Cluster Computing, 2005, Sep. 2005.
Humaira Kamal, et al.; Proceedings of the 2005 ACM/IEEE conference on Supercomputing, Seattle, p. 30, Washington, Nov. 12, 2005.
Sumitha Bhandarkar, et al.; ACM Computer Communication Review, vol. 36, No. 1, pp. 41-50, 2006, Jan. 2006.
H. K. Jerry Chu; Proceedings of the USENIX Annual Technical Conference 1996, Jan. 1996.
Ken Calvert; ACM Computer Communication Review, vol. 36, No. 2, pp. 27-30, 2006, Apr. 2006.
Jon Crowcroft; ACM Computer Communication Review, vol. 36, No. 2, pp. 51-52, 2006, Apr. 2006.
Greg Minshall, et al.; ACM Computer Communication Review, vol. 36, No. 3, pp. 79-92, 2006, Jul. 2006.
David Wetherall; ACM Computer Communication Review, vol. 36, No. 3, pp. 77-78, 2006, Jul. 2006.
Patrick Geoffray; HPCWire article: http://www.hpcwire.com/features/17886984.html, Aug. 18, 2006.
Geoffray P., "Protocol off-loading vs on-loading in high-performance networks," 14th Symposium on High Performance Interconnects, Aug. 23, 2006, 5pp.
Jose Carlos Sancho, et al.; Proceedings of the 2006 ACM/IEEE conference on Supercomputing, Tampa, Florida, Nov. 11, 2006.
Sayantan Sur, et al.; Proceedings of the 2006 ACM/IEEE conference on Supercomputing, Tampa, Florida, Nov. 11, 2006.
Steven Pope, David Riddoch; ACM Computer Communication Review, vol. 37, No. 2, pp. 89-92, 2007, Mar. 19, 2007.
Kieran Mansley, et al.; Euro-Par Conference 2007, pp. 224-233, Rennes, France, Aug. 28, 2007.
M. Kaiserswerth; IEEE/ACM Transactions in Networking vol. 1, Issue 6, pp. 650-663, 1993, Dec. 1993.
Danny Cohen, et al.; ACM Computer Communication Review, vol. 23, No. 4, p. 32-44, 1993, Jul. 1993.
J. Evans and T. Buller; IEEE TCGN Gigabit Networking Workshop, 2001, Apr. 22, 2001.
M.V. Wilkes and R.M. Needham; ACM Sigops Operating Systems Review, vol. 14, Issue 1, pp. 21-29, 1980, Jan. 1980.
Dickman, L., "Protocol OffLoading vs OnLoading in High Performance Networks," 14th Symposium on High Performance Interconnects, Aug. 23, 2006, 8pp.
Mogul J., "TCP offload is a dumb idea whose time has come," USENIX Assoc., Proceedings of HotOS IX: The 9th Workshop on Hot Topics in Operating Systems, May 2003, pp. 24-30.
Petrini F., "Protocol Off-loading vs On-loading in High-Performance Networks," 14th Symposium on High Performance Interconnects, Aug. 23, 2006, 4pp.
C. A. Thekkath, et al.; ACM Computer Communication Review, vol. 23, No. 4, 1993, Oct. 1993.
Raj K. Singh, et al.; Proceedings of the 1993 ACM/IEEE conference on Supercomputing, p. 452-461, Portland, Oregon, Nov. 15, 1993.
Peter Druschel and Larry L. Peterson; ACM Operating Systems Review, vol. 27, Issue 5, p. 189-202, 1993, Dec. 1993.
Matthias Kaiserswerth; IEEE/ACM Transactions on Networking, vol. 1, No. 6, p. 650-663, 1993, Dec. 1993.
Chris Maeda, Brian Bershad; ACM Operating Systems Review, vol. 27, Issue 5, p. 244-255, 1993, Dec. 1993.
Greg Regnier, et al.; IEEE Micro, vol. 24, No. 1, p. 24-31, 1994, Jan. 1994.
J. Vis; ACM Computer Communication Review, vol. 24, No. 1, pp. 7-11, 1994, Jan. 1994.

(56) **References Cited**

OTHER PUBLICATIONS

Danny Cohen, Gregory Finn, Robert Felderman, Annette DeSchon; Journal of High Speed Networks, Jan. 3, 1994.
Gregory G. Finn and Paul Mockapetris; Proceedings of InterOp '94, Las Vegas, Nevada, May 1994.
Stuart Wray, et al.; Proceedings of the International Conference on Multimedia Computing and Systems, p. 265-273, Boston, 1994, May 1994.
Various forum members; Message-Passing Interface Forum, University of Tennessee, Knoxville, 1994, May 5, 1994.
Raj K. Singh, et al.; ACM Computer Communication Review, vol. 24, No. 3, p. 8-17, 1994, Jul. 1994.
P. Druschel, et al.; ACM Computer Communication Review, vol. 24, No. 4, 1994, Oct. 1994.
Sally Floyd; ACM Computer Communication Review, vol. 24, No. 5, p. 8-23, 1994, Oct. 1994.
A. Edwards, et al.; ACM Computer Communication Review, vol. 24, No. 4, pp. 14-23, 1994, Oct. 1994.
L. S. Brakmo, et al.; ACM Computer Communication Review, vol. 24, No. 4, p. 24-35, 1994, Oct. 1994.
A. Romanow and S. Floyd; ACM Computer Communication Review, vol. 24, No. 4, p. 79-88, 1994, Oct. 1994.
R. J. Black, I. Leslie, and D. McAuley; ACM Computer Communication Review, vol. 24, No. 4, p. 158-167, 1994, Oct. 1994.
Babak Falsafi, et al.; Proceedings of the 1994 conference on Supercomputing, pp. 380-389, Washington D.C., Nov. 14, 1994.
Mengjou Lin, et al.; Proceedings of the 1994 conference on Supercomputing, Washington D.C., Nov. 14, 1994.
Nanette J. Boden, et al.; Draft of paper published in IEEE Micro, vol. 15, No. 1, pp. 29-36, 1995, Nov. 16, 1994.
Thomas Sterling, et al.; Proceedings of the 24th International Conference on Parallel Processing, pp. 11-14, Aug. 1995.
K. Kleinpaste, P. Steenkiste, B. Zill; ACM Computer Communication Review, vol. 25, No. 4, p. 87-98, 1995, Oct. 1995.
C. Partridge, J. Hughes, J. Stone; ACM Computer Communication Review, vol. 25, No. 4, p. 68-76, 1995, Oct. 1995.
A. Edwards, S. Muir; ACM Computer Communication Review, vol. 25, No. 4, 1995, Oct. 1995.
J. C. Mogul; ACM Computer Communication Review, vol. 25, No. 4, 1995, Oct. 1995.
Thorsten von Eicken, et al.; ACM Operating Systems Review, vol. 29, Issue 5, p. 109-126, 1995, Dec. 1995.
D. L. Tennenhouse, D. J. Wetherall; ACM Computer Communication Review, vol. 26, No. 2, pp. 15-20, 1996, Apr. 1996.
Paul Ronald Barham; PhD Thesis, University of Cambridge, 1996, Jul. 1996.
Chi-Chao Chang, et al.; Proceedings of the 1996 ACM/IEEE conference on Supercomputing, Pittsburgh, Nov. 17, 1996.
Joe Touch, et al.; "Atomic-2" slides, Gigabit Networking Workshop '97 Meeting, Kobe, Japan, Apr. 1997, 10pp.
Joe Touch, et al.; "Host-based Routing Using Peer DMA," Gigabit Networking Workshop '97 Meeting, Kobe, Japan, Apr. 1997, 2pp.
O. Angin, et al.; ACM Computer Communication Review, vol. 27, No. 3, pp. 100-117, 1997, Jul. 1997.
Charles P. Thacker and Lawrence C. Stewart; ACM Operating Systems Review, vol. 21, Issue 4, p. 164-172, 1987, Oct. 1997.
Ed Anderson, et al.; Proceedings of the 1997 ACM/IEEE conference on Supercomputing, p. 1-17, San Jose, California, Nov. 16, 1997.
Harvey J. Wassermann, et al.; Proceedings of the 1997 ACM/IEEE conference on Supercomputing, p. 1-11, San Jose, California, Nov. 16, 1997.
Philip Buonadonna, et al.; Proceedings of the 1998 ACM/IEEE conference on Supercomputing, p. 1-15, Orlando, Florida, Nov. 7, 1998.
Parry Husbands and James C. Hoe; Proceedings of the 1998 ACM/IEEE conference on Supercomputing, p. 1-15, Orlando, Florida, Nov. 7, 1998.
Michael S. Warren, et al.; Proceedings of the 1998 ACM/IEEE conference on Supercomputing, Orlando, Florida, Nov. 7, 1998.
John Salmon, et al.; Proceedings of the 1998 ACM/IEEE conference on Supercomputing, Orlando, Florida, Nov. 7, 1998.
Boon S. Ang, et al.; Proceedings of the 1998 ACM/IEEE conference on Supercomputing, Orlando, Florida, Nov. 7, 1998.
S. L. Pope, et al.; Parallel and Distributed Computing and Networks, Brisbane, Australia, 1998, Dec. 1998.
M. de Vivo, et al.; ACM Computer Communication Review, vol. 29, No. 1, pp. 81-85, 1999, Jan. 1999.
M. Allman; ACM Computer Communication Review, vol. 29, No. 3, 1999, Jul. 1999.
Steve Muir and Jonathan Smith; Technical Report MS-CIS-00-04, University of Pennsylvania, 2000, Jan. 2000.
Patrick Crowley, et al.; Proceedings of the 14th international conference on Supercomputing, pp. 54-65, Santa Fe, New Mexico, May 8, 2000.
Jonathan Stone, Craig Partridge; ACM Computer Communication Review, vol. 30, No. 4, pp. 309-319, 2000, Oct. 2000.
W. Feng and P. Tinnakornsrisuphap; Proceedings of the 2000 ACM/IEEE conference on Supercomputing, Dallas, Texas, Nov. 4, 2000.
Jenwei Hsieh, et al.; Proceedings of the 2000 ACM/IEEE conference on Supercomputing, Dallas, Texas, Nov. 4, 2000.
Ian Pratt and Keir Fraser; Proceedings of IEEE Infocom 2001, pp. 67-76, Apr. 22, 2001.
"Nvidia Tesla GPUs to Communicate Faster Over Mellanox InfiniBand Networks," press release dated Nov. 25, 2009, Portland OR, 3 pp: <http://gpgpu.org/2009/11/25/nvidia-tesla-mellanox-infiniband>.
"Nvidia GPUDirect™ Technology—Accelerating GPU-based Systems," Mellanox Tech. Brief, May 2010, 2pp: <http://www.mellanox.com/pdf/whitepapers/TB_GPU_Direct.pdf>.
Pope, S.L. et al., "Enhancing Distributed Systems with Low-Latency Netowrking," Olivetti and Oracle Research Laboratory, Cambridge Univ. May 1998, 12 pp: <http://www.cl.cam.ac.uk/research/dtg/www/publications/public/files/tr.98.6.pdf>.
Hodges, S.J. et al., "Remoting Peripherals using Memory-Mapped Networks," Olivetti and Oracle Research Laboratory, Cambridge Univ., 1998, 3 pp: <http://www.cl.cam.ac.uk/research/dtg/www/publications/public/files/tr.98.7.pdf>.
U.S. Appl. No. 12/964,642—Notice of Allowance dated Nov. 26, 2014, 21 pages.
U.S. Appl. No. 13/624,788—Notice of Allowance dated Dec. 5, 2014, 12 pages.

* cited by examiner

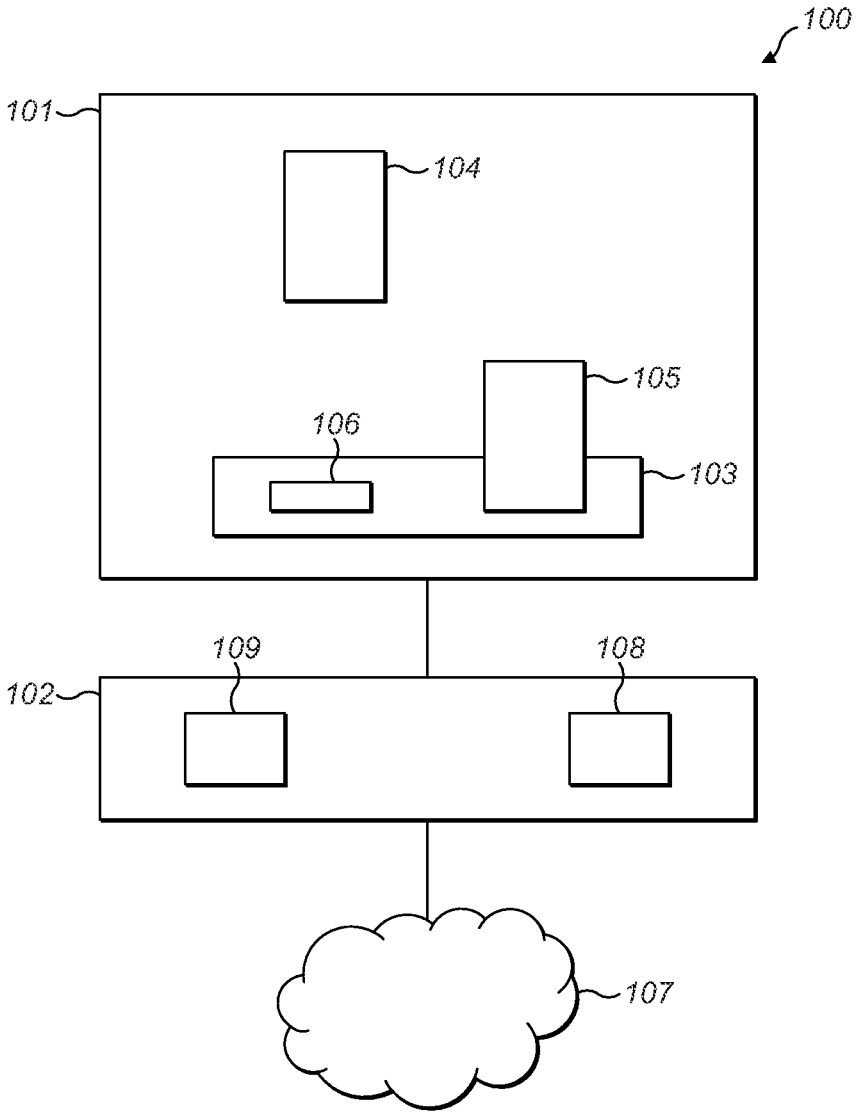


FIG. 1

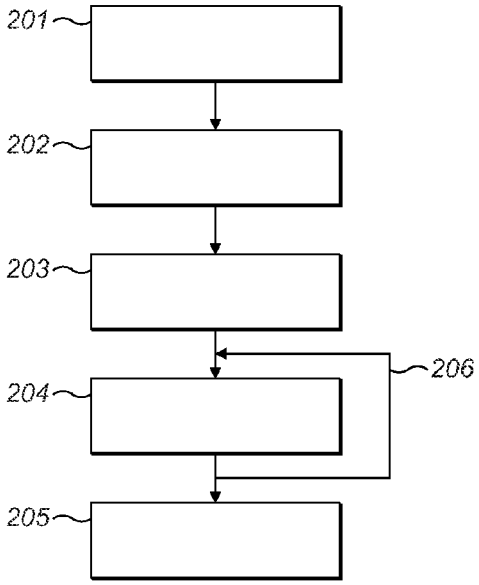

201
202
203
204
205
206

FIG. 2

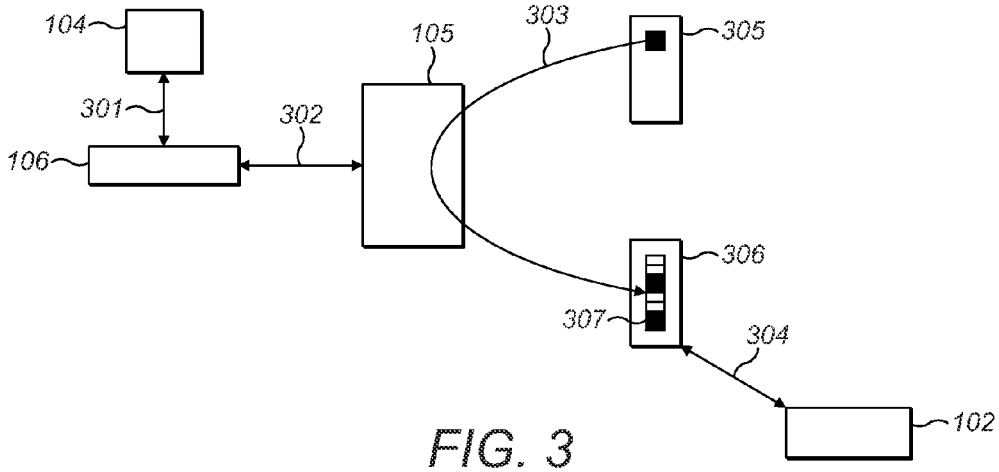

104
301
106
302
105
303
305
306
307
304
102

FIG. 3

REDUCING NETWORK LATENCY

BACKGROUND

This invention relates to low-latency methods for transmitting data at a data processing system.

It is generally desirable to minimise the latency associated with sending messages over a network between applications. This enables a receiving application to receive a message the minimum possible time after the sending application forms the message.

It is especially important to minimise the latency over networks that connect high performance computing devices, or computing devices that must react as quickly as possible to incoming data in order to gain a commercial advantage, such as electronic trading devices. In electronic markets, messages sent over networks are used to submit orders and quotes to exchanges and it is often of considerable benefit for a computing system to be able to respond to external stimuli and submit orders and quotes ahead of competitors.

Applications running at computing devices on a network typically communicate over the network using an application-level protocol (such as HTTP or the Financial Information Exchange or FIX protocol) that define a series of structured messages. In order to form each message, the data for transmission must be processed in accordance with the application-level protocol (as well as lower layer protocols, such as Ethernet) in order to form a completed message that is ready for transmission over the network. Typically this protocol processing takes the form of generating headers and calculating error checks (such as CRCs). Such protocol processing can introduce significant latency into the transmission of data since in conventional data transmission systems a network protocol stack must wait for all of the data that is to form a message to be available before commencing protocol processing. This can significantly extend the time elapsed between an application determining that a message is to be sent and that message actually being transmitted onto the wire.

Other causes of transmission latency in conventional systems include the restriction for connection-oriented protocols (such as TCP) that only the protocol stack that negotiated the connection can transmit over the connection. Often such a protocol stack is supported at a host kernel, which does not present a low latency transmission path to applications of the system. Additionally, a modern processor can introduce significant latency if the state and instructions required by a transmission process is not available in cache at the processor.

There is therefore a need for improved methods for transmitting data at a data processing system that reduce the latency associated with message transmission.

SUMMARY

Roughly described, according to a first aspect of the first invention there is provided a method of transmitting data for use at a data processing system supporting an application, a software interface and a network protocol stack, the data processing system being coupled to a network by a network interface device, the method comprising: the application: forming one or more parts of a message for transmission; causing a message template to be generated by means of a first request to the software interface; the software interface: on receiving the first request, causing the network protocol stack to include said one or more parts of the message for transmission in the message template and, in accordance with a predetermined set of protocols, form at least in part one or more protocol headers for the message for transmission; and subsequently: forming one or more further parts of the message for transmission; causing the network protocol stack to include the one or more further parts of the message in the message template and, on completion of the message for transmission, process the message in accordance with the predetermined set of protocols so as to complete the protocol headers; and causing the network interface device to transmit the completed message over the network.

Suitably the step of forming one or more further parts of the message for transmission is performed by the application and the step of causing the network protocol stack to include the one or more further parts of the message in the message template is effected by means of a second request made by the application to the software interface. Suitably the application makes a series of second requests to the software interface, each second request being made on one or more further parts of the message for transmission becoming available to the application. The final request of the second request type in the series preferably completes the message for transmission, causing the network protocol stack to process the message in accordance with the predetermined set of protocols so as to complete the protocol headers and the network protocol stack to transmit the completed message.

Preferably the network protocol stack comprises at least two parts: a first part supported at the data processing system and a second part supported at the network interface device; the first part being configured to perform the step of forming at least in part one or more protocol headers of the message for transmission, and the second part being configured to perform the step of processing the message so as to complete the protocol headers. Preferably the step of forming the one or more further parts of the message for transmission is performed at a data processing engine of the network interface device. Preferably the step of causing the second part of the network protocol stack to process the message so as to complete the protocol headers is performed in response to completing formation of the one or more further parts of the message for transmission.

Preferably the first part of the network protocol stack is a user-level protocol stack.

Suitably at least part of the message template is held at the network interface device.

Preferably the first request includes memory references to the one or more parts of a message for transmission.

The method suitably further comprises, subsequent to the network protocol stack including said one or more parts of the message for transmission in the message template, the network protocol stack performing segmentation of the message template in accordance with a predetermined transport protocol such that the one or more parts of the message are transmitted over the network in one or more message segments by the network interface device. The method suitably further comprises, subsequent to the network protocol stack including said one or more further parts of the message for transmission in the message template, the network protocol stack causing those segments of the message template that correspond to the further parts of the message for transmission to be transmitted over the network by the network interface device. Preferably the step of the network protocol stack causing those segments of the message template that correspond to the further parts of the message for transmission to be transmitted over the network completes the step of the network interface device transmitting the message over the network.

The predetermined transport protocol could be the transmission control protocol.

Suitably the transport protocol is configured such that a message segment can carry a plurality of parts of the message for transmission that are not contiguous in the message for transmission. Suitably a message segment can include one or more header fields that determine the logical position within the message for transmission of the parts of the message in the message segment. Suitably the one or more header fields are formed in a TCP header option. Preferably all of the one or more parts of the message are transmitted over the network in a single message segment by the network interface device.

Suitably at least part of the message template is stored at the network interface device, that part of the message template stored at the network interface device including the one or more parts of a message for transmission.

Preferably the step of the network protocol stack including the one or more further parts of the message in the message template comprises transferring the one or more further parts of the message into the message template using one or more programmed I/O instructions. Alternatively the step of the network protocol stack including the one or more further parts of the message in the message template comprises: for those further parts of the message that are below a predetermined length, transferring the one or more further parts of the message into the message template using one or more programmed I/O instructions; and for those further parts of the message that are above the predetermined length, transferring the one or more further parts of the message into the message template by means of direct memory access by the network interface device.

Preferably the software interface is configured to provide a dummy transmission function by means of which an application can cause the network protocol stack to process the message template in accordance with the predetermined set of protocols as if for transmission by the network interface device, the method further comprising, prior to the step of the network interface device transmitting the message over the network, the application periodically calling the dummy transmission function so as to cause the network protocol stack to process the message template in accordance with the predetermined set of protocols without causing the network interface device to transmit the processed message template over the network.

Suitably the step of calling the dummy transmission function comprises, prior to the network protocol stack processing the message template in accordance with the predetermined set of protocols, the network protocol stack padding the message template with data so as to fill the missing parts of the message for transmission.

The message template could be a logical expression of the message for transmission.

Preferably the software interface is an application programming interface.

Preferably at least part of the network protocol stack is at user level.

According to a second aspect of the first invention there is provided a data processing system coupled to a network by a network interface device and comprising: an application configured to form one or more parts of a message for transmission; a network protocol stack configured to process messages for transmission in accordance with a predetermined set of network protocols; a software interface configured to, in response to a first request, cause the network protocol stack to form at least in part one or more protocol headers of a message for transmission; wherein the application is configured to, on forming one or more parts of a message for transmission, make a first request to the software interface so as to cause the network protocol stack to form a message template comprising at least in part one or more protocol headers of a message for transmission and the one or more parts of the message for transmission; and subsequently: the network protocol stack being configured to, on one or more further parts of the message for transmission being formed, include the one or more further parts of the message in the message template and, on completion of the message for transmission, process the message in accordance with the predetermined set of protocols so as to complete the protocol headers and cause the network interface device to transmit the completed message over the network.

Suitably the application is configured to form the one or more further parts of the message for transmission and to cause the network protocol stack to include the one or more further parts of the message in the message template by means of a second request to the software interface.

Preferably the one or more further parts of the message for transmission are formed at the network interface device at a data processing engine of the network interface, and the network protocol stack comprises at least two parts: a first part supported at the data processing system and a second part supported at the network interface device; the first part being configured to perform the step of forming at least in part one or more protocol headers of the message for transmission, and the second part being configured to perform the step of processing the message so as to complete the protocol headers.

According to a third aspect of the first invention there is provided a method of transmitting data for use at a data processing system and network interface device, the data processing system being coupled to a network by the network interface device, the method comprising: forming a message template in accordance with a predetermined set of network protocols, the message template including at least in part one or more protocol headers; forming an application layer message in one or more parts; updating the message template with the parts of the application layer message; processing the message template in accordance with the predetermined set of network protocols so as to complete the protocol headers; and causing the network interface device to transmit the completed message over the network.

Preferably the step of forming a message template is performed at a network protocol stack supported at the data processing system. The step of forming an application layer message could be performed by one of an application supported at the data processing system, and a data processing engine of the network interface device or other peripheral device of the data processing system. Preferably the processing step is performed at a packet processing engine supported at the network interface device.

According to a fourth aspect of the first invention there is provided a data processing system coupled to a network, the data processing system comprising: a network protocol stack configured to, in accordance with a predetermined set of network protocols, form a message template including at least in part one or more protocol headers; and a message forming engine configured to form an application layer message in one or more parts and, on the formation of each of those parts, cause the respective part of the message to be applied to the message template; wherein the network protocol stack is further configured to, on the application layer message being completed, process the message template in accordance with the predetermined set of network protocols so as to complete the protocol headers and cause the completed message to be transmitted over the network.

Preferably the network protocol stack is supported at a network interface device arranged to couple the data processing system to the network. The message forming engine could

be one or both of an application supported at the data processing system or a data processing engine at a network interface device arranged to couple the data processing system to the network.

According to a first aspect of the second invention there is provided a method of transmitting data in accordance with a connection-oriented network protocol, the method being for use at a data processing system coupled to a network and supporting first and second packet processing engines, the method comprising: initiating a connection at the first packet processing engine; the first packet processing engine passing first network protocol state to the second packet processing engine sufficient to permit the second packet processing engine to transmit a specified amount of data over the connection; forming application layer data for transmission; and the second packet processing engine performing packet processing of the application layer data in dependence on the first network protocol state so as to form one or more network messages and causing transmission of one or more network messages over the connection.

Preferably the data processing system is coupled to the network by a network interface device and the second packet processing engine is supported at the network interface device.

Preferably the step of forming application layer data for transmission is performed at a data processing engine of the network interface device. Alternatively the step of forming application layer data for transmission is performed at an application supported at the data processing system. Alternatively the step of forming application layer data for transmission is performed at a data processing engine supported at a peripheral device of the system.

Preferably the passing step is performed in response to a request made by the second packet processing engine.

The first network protocol state could be a message template including at least in part one or more protocol headers for a network message.

Preferably the first packet processing engine performs processing of network messages received over the connection. Preferably the first packet processing engine is a software network protocol stack supported at kernel or user level at the data processing system. Preferably the second packet processing engine is configured to perform packet processing of the application layer data using an FPGA, ASIC, dedicated hardware, or embedded software.

The method could further comprise, subsequent to the passing step, the first packet processing engine signalling to the second packet processing engine so as to cause the second packet processing engine to return control of the connection to the first packet processing engine. The method could further comprise, in response to the signalling, the second packet processing engine completing any pending transmission operations over the connection and passing second network protocol state to the first packet processing engine so as to return control of the connection to the first packet processing engine.

The method preferably further comprises, on the amount of data transmitted over the connection becoming within a predetermined amount of the specified amount of data, the second packet processing engine requesting updated network protocol state from the first packet processing engine and, in response to the request, the first packet processing engine passing updated network protocol state to the second packet processing engine. The method preferably further comprises: forming additional application layer data for transmission; and the second packet processing engine performing packet processing of the additional application layer data in dependence on the updated network protocol state so as to form one or more network messages and causing transmission of the one or more network messages over the connection.

Preferably the steps of forming the application layer data and performing packet processing comprise: forming one or more first parts of the application layer data; the second packet processing engine processing the one or more parts of the application layer data in dependence on the first network protocol state so as to form a message template including at least in part one or more protocol headers; and subsequently: forming one or more additional parts of the application layer data; and updating the message template with the one or more additional parts of the application layer data so as to complete the message template and processing the message template so as to complete the protocol headers.

Preferably the method further comprises, once the specified amount of data has been transmitted over the connection, the second packet processing engine returning control of the connection to the first packet processing engine.

Suitably the connection-oriented network protocol is TCP.

In any aspect of any of the inventions disclosed herein a data processing engine could be an FPGA, ASIC, dedicated hardware, or embedded software.

According to a second aspect of the second invention there is provided a data processing system coupled to a network, the data processing system comprising: a message forming engine operable to form application layer data for transmission over the network in accordance with a connection-oriented network protocol; a first packet processing engine operable to initiate a connection in accordance with the connection-oriented network protocol; a second packet processing engine operable to perform packet processing of application layer data so as to form one or more network messages in accordance with the connection-oriented network protocol; wherein the first packet processing engine is configured to, subsequent to initiating the connection, pass first network protocol state to the second packet processing engine sufficient to permit the second packet processing engine to transmit a specified amount of data over the connection, and the second packet processing engine is configured to, on receiving application layer data from the message forming engine, perform packet processing of the application layer data in dependence on the first network protocol state so as to form one or more network messages and cause transmission of one or more network messages over the connection.

According to a first aspect of a third invention there is provided a method for minimising execution latency of a communication process provided for execution at a data processing system, the data processing system being coupled to a network and supporting a software interface, an application and a network protocol stack providing a communication process, and the method comprising the application periodically making a call to a dummy communication function of the software interface so as to cause the network protocol stack to perform the communication process without communicating data over the network.

Preferably the software interface provides a communication function corresponding to the dummy communication function and the method further comprises the application subsequently making a call to the communication function in respect of a network message so as to cause the network protocol stack to perform the communication process and communicate the network message over the network. Suitably the dummy communication function and the corresponding communication function are one and the same, and

the dummy communication function is identified by means of a flag of the communication function.

Suitably the communication process is a transmit process and the step of making a call to a communication function comprises the application periodically making a call to a dummy transmit function of the software interface so as to cause the network protocol stack to perform the transmit process without transmitting data over the network.

Suitably the dummy transmit function is called in respect of a partial message held at a transmit buffer of the data processing system.

Suitably the communication process is a receive process and the step of making a call to a communication function comprises the application periodically making a call to a dummy receive function of the software interface so as to cause the network protocol stack to perform the receive process of the network protocol stack without receiving data over the network.

DESCRIPTION OF THE DRAWINGS

The present invention will now be described by way of example with reference to the accompanying drawings, in which:

FIG. **1** is a schematic diagram of a data processing system and network interface device configured in accordance with the present invention.

FIG. **2** is a flow chart illustrating data transmission in accordance with the present invention.

FIG. **3** shows a general overview of the interaction of the various components of FIG. **1**.

DETAILED DESCRIPTION

The following description is presented to enable any person skilled in the art to make and use the inventions, and is provided in the context of a particular application. Various modifications to the disclosed embodiments will be readily apparent to those skilled in the art.

The general principles defined herein may be applied to other embodiments and applications without departing from the spirit and scope of the present inventions. Thus, the present inventions are not intended to be limited to the embodiments shown, but is to be accorded the widest scope consistent with the principles and features disclosed herein.

The present inventions relate to the transmission of data over a network between data processing systems. A data processing system could be any kind of computing device, such as a server, personal computer or handheld device. The present inventions are described herein by way of example with reference to networks that operate TCP/IP over Ethernet, but it will be appreciated by the skilled person that the present invention is not so limited and could be implemented with any kind of network (wired or wireless) and set of network protocols. The present inventions are particularly suitable for use at a data processing system and network interface device (NIC)_configured such that at least part of the protocol processing of data packets for transmission is performed at the NIC. Thus, at least some of the network protocol stack can be supported at the NIC, with one or more network protocols being offloaded in part or in full to the NIC to be performed in hardware at the NIC.

The present inventions address the latencies introduced in the transmission of data packets or messages over a network when the data for some parts of a network message are not immediately known to the entity that forms the application-layer message and requests the data transmission operation. Often, data that is to constitute some parts of a message are immediately known to an application on determining that a message is to be sent, whilst data for other parts of the message are not known until just before the message is transmitted over the network. In conventional transmission systems, the network stack waits for all of the data for the message to become available prior to performing protocol processing and transmitting the message.

The first of the inventions provides a mechanism by which a network message can be incrementally constructed by one or more network protocol stacks and one or more entities forming application layer data. This can be achieved through the provision of an interface by which an application supported at a data processing system can provide data for transmission to a network stack as and when the data becomes available, allowing the network stack to perform any possible protocol processing and minimising the latency caused by data that is available only at the last moment. The second of the inventions provides a mechanism by which a stream of a connection-oriented protocol can be handed between network protocol stacks. This allows a host to manage the protocol streams but permit another network protocol stack to perform at least some transmission operations at low latency. It is intended that the first and second inventions can be used together. Thus the first invention along with any of its optional features can be implemented with the second invention along with any of its optional features. The first and second inventions described below are therefore not mutually exclusive and disclosure is made of the combination of the first invention in any of its embodiments and the second invention in any of its embodiments.

A schematic diagram of a system **100** configured in accordance with the present inventions is shown in FIG. **1**. Data processing system **101** supports an operating system **103** and an application **104** that is operable to communicate over network **107** by means of network interface device or NIC **102** and network protocol stack **105**. The network protocol stack is illustrated as a logical block in the figure and all or part of the stack could be supported at kernel, user-level, or at the network interface device **102**. All or part of the protocols of the network stack could be performed at a packet processing engine **108** of the network interface device itself. The network interface device could optionally also support a message generating entity **109** configured to generate application-layer network messages in an analogous manner to the way in which application **104** might form messages for transmission over the network. The network interface device could be supported at the data processing system; for example, it could be provided at the mainboard of the data processing system.

The first invention will now be described by way of example with reference to FIG. **1**.

The first invention provides a mechanism by which a network message can be incrementally constructed by one or more network protocol stacks and one or more entities forming application layer data. On a request being made to transmit a network message (for example, by means of a call from an application to a socket API provided by the operating system), a network protocol stack of system **100** is configured to generate a message template for the network message in accordance with a predetermined set of network protocols such that the message template includes at least in part one or more protocol headers for the message. If, at the time the transmit request is made, any data for transmission is available from one or more entities of the system that are arranged to form application layer data, the network protocol stack also includes that data in the message template.

Thus, a message template that can include parts of the application-layer message for transmission is generated whilst the remainder of the message is formed. As further parts of the message become available, the message template can be updated with those parts of the message. Once all of the message is available, a network protocol stack of the system completes protocol processing of the data packet so as to complete its protocol headers and the packet is transmitted over the network by the network interface device. The message template is a logical construct that includes the known parts of the message and headers (and/or other data) generated by the protocol stack. The data defining a message template could be included by means of one or more memory references (e.g. to the known parts of the message). The message template could be contiguously defined in memory such as a transmit buffer.

Any of application **104**, operating system **103**, message generating entity **109** at the NIC, or a message generating entity at another peripheral device of system **100** could be an entity forming application layer data. Either the kernel or user-level network stack **105**, or packet processing engine **108** at the NIC represent network protocol stacks. Sometimes it is advantageous for one entity forming application layer data to request transmission of a message and optionally provide some of the message data, and for another entity forming application layer data to complete the message. The first invention can be performed and a network data packet built up incrementally irrespective of where the protocol processing is performed and where the message data is formed.

Similarly, it can be advantageous for one network protocol stack to commence packet processing and for another network protocol stack to complete packet processing. For example, application **104** at the data processing system could request transmission of a network message and provide some of the data for that message before passing over to network generating entity **109** at the NIC, which generates the remainder of the message. The message template in this example be generated at a user level network stack **105** before passing control of the remainder of the processing of the network message to packet processing engine **108** at the NIC. This example ensures that the final parts of a message and the completion of protocol processing of the network message is performed with low latency at the NIC.

According to a preferred embodiment of the first invention, a software interface **106** is provided by means of which application **104** at the host data processing system can request that data for transmission is formed into a message by network stack **105** and transmitted over network **107** by means of NIC **102**. Preferably, software interface **106** is an application programming interface or API.

A flow chart illustrating data transmission in accordance with the present invention is shown in FIG. **2**. On determining that data is to be sent over a network to a remote endpoint, at step **201** application **104** forms those parts of the message that are known to the application and at step **202** requests by means of software interface **106** that a message template is created. The application passes a reference to the data constituting the known parts of the message in its request **202**, typically a memory reference to the location of the those message parts in an application buffer. In response to the request the software interface causes network protocol stack **105** to create a message template at step **203** including those known parts of the message that were formed by the application at step **201**.

Preferably the network stack populates the message template with data by copying the data known to the application from the application's buffers to the transmit buffer holding the message template, with the data being written to the template at the position that data is to have in the completed message. Alternatively, references to the known application data are included in the message template so as to allow the full message to be logically read from the transmit buffer when the message is completed. The message template represents the completed message once it includes all data for transmission from the application and all protocol processing has been completed by the protocol stack (e.g. to form the appropriate headers for the message). Protocol processing may be performed in software at the data processing system, or in hardware at the network interface device. Typically some protocols are completed on-the-fly by the network interface device as the message traverses the NIC—for example, the calculation of CRC check data for Ethernet data packets. A completed message is transmitted over the network at step **205**.

Software interface **106** is configured so as to provide a mechanism by which an application can provide further parts of the message to the network stack as they become known to the application. This is shown by message update step **204** in FIG. **2**, which comprises the application making a request to the software interface by means of which it passes one or more references to the data that is to be included as further parts of the message. In response to the message update request, software interface **106** causes the network protocol stack to include the newly-available data into the message template. Message update step **204** can be performed as many times as is necessary to complete the message template, as indicated by return arrow **206**.

A general overview of the interaction of the various components of FIG. **1** is shown in FIG. **3**. Application **104** is configured to access software interface **106** by means of a set of commands or requests **301**. Preferably software interface **106** is an application programming interface (API) with requests **301** being calls to the API. As indicated by arrow **302**, the software interface is configured to cause network protocol stack **105** to include data held at the application buffer(s) **305** of application **104** into message template **307** held in transmit buffer **306**. Arrow **303** indicates the transfer of data or references to data from the application buffers into the message template.

Additionally, network stack **105** is configured to perform on the data of the message template the appropriate possible protocol processing, such as the formation of protocol headers and footers. For example, even when none or only some of the data of the message is known, for many protocols it is possible to form the source and destination addresses and specify certain header options, such as packet length and sequence number. Performing this processing whilst the application is waiting for some of the packet data to become available minimises the latency associated with transmitting the message since the amount of protocol processing to be performed once all of the message data is known is reduced and hence the time between the last of the message data being available and the point at which the message is actually transmitted is reduced.

Often most of the payload data of a data packet will be available to an application, but some parts of the payload data will only be available immediately prior to the transmission of the data packet over a network. This is typically because the application is performing calculations to determine those parts of the message data. For example, in the case in which the application is an electronic trading application, the application might only determine the price, quantity or symbol for a trade at the last moment before an electronic trading message is sent. However, it is vitally important that the latency of

the trade is as small as possible and the present invention achieves this by arranging that as much of the protocol processing of the data packet as possible is performed prior to the final parts of the message being available.

The message template may be held in transmit buffers at the data processing system or at the NIC, or the logical message template could be held in part in memory at the data processing system and in part at memory at the NIC. If the message template **307** is held at the data processing system, the network interface device **102** is preferably configured to have direct memory access (DMA) to transmit buffer(s) **306** and hence reads completed messages from the transmit buffers of the data processing system by means of DMA (indicated by arrow **304** in FIG. **3**). If the message template **307** is held in transmit buffer(s) **306** at the NIC, the NIC **102** can access (arrow **304**) the message template directly. Arranging that the message template is held at the NIC, with at least some further protocol processing being performed at the NIC can help to further minimise the latency of transmission operations.

Note that network protocol stack **105** is illustrated in the figures as a single entity but may in fact comprise one or more software entities that could be located at kernel or user level at the data processing system. The stack **105** could be a user-level protocol stack configured to perform protocol processing in the context of user-level software so as to minimise context switches into the kernel. Further parts of the total network protocol stack could be performed at the network interface device **102**. For example, in the case of a network interface device that supports TCP offload, the majority of TCP protocol processing would be performed at the network interface device.

Protocol processing could be performed at NIC **102** at a packet processing engine **109**, which could be, for example, an FPGA, ASIC, embedded software or a hardware protocol processing engine configured to perform dedicated processing according to one or more network protocols. In certain configurations, it can be advantageous for a data processing system and NIC to support multiple entities capable of performing protocol processing. For example, a data processing system might support both kernel and user-level protocol stacks, and at least part of a stack supported at the data processing system could also be supported at the NIC. The data processing system would typically be configured to select the appropriate stack in dependence on the message type, network conditions, or other factors.

Preferably, the initial protocol processing associated with forming the message template at step **203** is performed in software at the data processing system and subsequent protocol processing associated with message update step **204** is performed at NIC **102**. This can have significant advantages when the missing parts of the message for transmission are provided as a result of processing performed at the NIC.

For example, consider the situation in which data processing system **101** and NIC **102** form an electronic trading platform whose operation will now be described with respect to the figures. At step **201**, the electronic trading software determines that a trade is to be placed at an electronic exchange accessible over network **107** and forms data for transmission in a bid message to the electronic exchange. At step **202**, the trading software requests by means of interface **106** that a message template be created and that protocol processing is initiated at a software protocol stack of the data processing system. In response, interface **106** causes the software protocol stack to commence processing of the message and generate the message template in a buffer memory at the NIC.

In the present example, most of the data for transmission is known to the application at the point at which creation of the message template is requested, with the missing parts of the message being determined at the NIC by means of trading algorithms running at the NIC at a message forming entity **108**—such as at a dedicated FPGA or ASIC, or by means of software running at a processor supported at the NIC. This allows the software protocol stack to substantially form the message template, leaving the NIC to add the results of its processing and complete the headers immediately prior to transmitting the message over the network from its buffer to the electronic exchange. By locating the final processing of the message as close as possible to the network at the NIC, latency is minimised.

Under the control of the trading software **104**, the trading algorithms running at the NIC determine the data for the missing parts of the message template whilst the protocol processing of step **203** is performed. As soon as the parameters of the trade that represent the final parts of the message have been determined by the algorithms, the missing parts of the message are at step **204** included in the message template by the NIC and the network protocol stack at the NIC (packet processing engine **109**) completes protocol processing of the data packet before at step **205** transmitting the message over the network to the electronic exchange. Preferably the processes running at the NIC cause the network protocol stack to complete protocol processing of the message. Preferably the processes are executed at one or more FPGA processors of the NIC, the one or more processors optionally including a memory at which the message template is stored.

Typically the trade message would be formed in accordance with the FIX application layer protocol over TCP/IP and Ethernet, with the trading software forming a message in accordance with the FIX protocol and the lower layer protocol stacks being generated at the software stack (which could be at user level) of the data processing system. This would leave the NIC to complete the payload data and message headers in accordance with the appropriate protocols once the trading algorithms at the NIC determine the parameters of the bid (e.g. number of stocks, stock symbol, etc.).

Three transmission mechanisms for use in accordance with the first invention will now be described.

A first transmission mechanism can be used with messaging protocols that support segmentation, such as the Transmission Control Protocol (TCP) which will now be discussed by way of example. TCP includes a segmentation algorithm that allows a stream of messages to be split into multiple segments, each of which includes a sequence number so that the receiver can reassemble the message stream even when segments arrive out of order. In this embodiment, the network protocol stack is configured to cause the network interface device to transmit at least some of the known parts of a message prior to the message template being completed. This is achieved by configuring the network protocol stack to send the known parts of a message template as TCP segments, with subsequent parts of the message being transmitted as further TCP segments as they become known.

For example, for a message of total length 30 bytes, with 2 bytes initially unknown at offset 10 bytes, the following TCP segments would be initially transmitted:

tcp-seq-1: seq=0, len=10

tcp-seg-2: seq=12, len=18

On receiving these segments, the receiving data processing system can deliver the first 10 bytes of the message to the receiving application but not any more since it does not yet

have the data at sequence position **10**. Later, when the missing 2 bytes are known, the following TCP segment would be transmitted:

tcp-seg-3: seq=10, len=2

The receiving data processing system can then deliver the remaining 20 bytes of the message to the receiving application.

The protocol processing stack is configured to perform segmentation of the incomplete message template by forming each contiguous known part of the message into a message segment and passing the segment to the network interface device for transmission. The protocol processing stack could be configured to only form a message segment for contiguous parts of the message that exceed a predetermined length so as to avoid the overhead associated with sending many small data packets. Note that the message template and message segment(s) need not be a contiguously stored at a transmit buffer and could be logically represented at a transmit buffer such that it is not necessary to duplicate known parts of the message data in memory—one or both of the message template and a message segment could refer to the location of the message data held in the transmit buffer.

This embodiment is of particular benefit if the known parts of the message are large and the updates are small, since the latency critical step of forming and sending the last parts of the segmented message involves sending one (or sometimes a small number of) small message segment(s).

However, there are problems with using the TCP protocol in this manner:

Each TCP segment can only contain a subset of the message that is contiguous in the sequence space of the message stream. Hence in the above example, two segments were needed to transfer only 28 bytes, which is highly inefficient. This is particularly a problem if a message template is completed by multiple discrete message parts that require multiple message segments to be transmitted in order to complete the transfer of the message data to the receiving data processing system because the formation and transmission of the last message segments is critical to the overall latency of the message transmission operation.

The technique can only be used if the size of the unknown segments of data are known in advance because it is necessary to know where each message part is located in the sequence space.

The receiving TCP implementation receives the arriving updates out of order and therefore will respond to such message segments with an ACK message. This can increase the latency of message reception.

In a second transmission mechanism it is therefore proposed that the TCP protocol is extended or replaced with a new transport protocol that allows multiple message segments to be delivered in a single multi-segment network message. This can be achieved by defining in the protocol a multi-segment message format arranged to carry multiple segments of a message as a contiguous data payload and header fields that specify the how the contiguous data payload should be split up into its constituent message segments at the receiving data processing system. For example, by specifying in header fields the byte offset within the parent message and length of each message segment in the payload, the message segments can be extracted from the payload of the multi-segment message and reassembled to form the parent message. The message segments forming the payload of the multi-segment message could be separated by markers delineating the message segments; these markers could be headers carrying the byte offset and optionally length information of the respective segment in the parent message.

The TCP protocol can be extended to support multiple message segments in each TCP segment by defining a new TCP header option type to describe the layout of the message segments.

For small messages, the cost of segmentation and reassembly in order to transfer known parts of a message in advance can outweigh the benefits. A third transmission mechanism proposes an alternative strategy in which known parts of a message are transferred to buffers at a network interface device in advance of those parts of a message not yet available to an application. In this embodiment, transmit buffer **306** is located in memory at the network interface device such that the message template is formed at a network interface device buffer and data for transmission is copied into the message template at the NIC in accordance with steps **203** and **204** of FIG. **2**. This ensures that as much data as possible is held as close as possible to the NIC at the point when the last of the message data becomes available.

Message data constituting the parts of the message to be held in the message template can be transferred to the NIC **102** using programmed I/O (PIO) instructions or direct memory access (DMA) by the NIC to buffers held at the data processing system (these buffers could include application buffer **305**, or be an intermediate buffer supported by the network protocol stack). Preferably message update step **204** in FIG. **2** is performed using PIO instructions to transfer newly-available message parts into the message template, with the host CPU of the data processing system performing the transfer. This has lower latency than DMA when small amounts of data are transferred. The initial transfer of data at step **203** from the application buffer to the transmit buffer can be performed by DMA.

Most preferably, the decision as to whether to perform message update step **204** by PIO or DMA transfer is made dynamically in dependence on the size of the message part being transferred, with messages below a predetermined threshold size being transmitted using PIO and messages above that threshold being transmitted using DMA.

The second of the inventions will now be described, which provides a mechanism by which a stream of a connection-oriented protocol can be handed between network protocol stacks.

In general it is not possible for more than one stack to transmit messages over a given connection of a connection-oriented network protocol such as TCP. In order to maintain the order of messages over a connection under varying network conditions, the state information (such as sequence number and receive window parameters) associated with that connection is required and that state information resides with the protocol stack that established the connection.

This invention provides a mechanism by which one network stack can hand over the ability to transmit over a connection to another network stack for a limited period. This will be illustrated by way of example with reference to the system **100** of FIG. **1**. Consider a first packet processing engine (network stack **105** at data processing system **101**) that has established a TCP connection over network **107** and a second packet processing engine (network stack **109** at network interface device **102**) that wishes to transmit over that connection. In accordance with the invention, network stack **105** passes network protocol state to the NIC network stack **109** that is sufficient to allow the second packet processing engine to transmit a specified amount of data over the connection. This state can include one or more sequence numbers, congestion window and/or receive window parameters, network addresses and port numbers. By transmitting this

state to network stack **109**, network stack **105** passes permission to the network stack **109** to transmit data over the network.

Typically the state includes at least the current sequence number and a parameter identifying the receive window remaining. This allows network stack **109** to start transmitting data packets over the connection until the remaining receive window is used up, at which point network stack **109** returns control to the parent network stack **105**. Note that the connection state itself need not be copied between the network stacks and the passing of connection state to network stack **109** could be achieved by passing one or more memory references to the state or a copy of the state held in memory at the data processing system.

Most generally, this invention provides a mechanism by which one network stack can hand over the ability to transmit over a connection to another network stack irrespective of the locations of those network stacks: either one could be at the kernel of operating system **103**, supported at user level of the data processing system, at packet processing engine **108** of NIC **102**, or at another peripheral device of the system.

Application layer data for transmission over the connection could be formed at any point in system **100**: at the kernel of operating system **103**, at application **104**, at message forming entity **109** of NIC **102**, or at another peripheral device of the system. Application layer data for transmission over the connection by means of the slave network stack (e.g. **109** in the above example) is provided to the slave network stack and the slave network stack is configured to process the application layer message in dependence on the received state so as to prepare the message for transmission over the connection by the NIC.

The second invention is particularly advantageous if the message forming entity is close to the slave network stack so as to minimise the latency between the application layer message being formed and the resulting data packet being sent over the connection. Most preferably, the slave network stack is packet processing engine **108** and the message forming entity **109** is located at the NIC. This arrangement is particularly effective at minimising the latency of transmission because the formation of application layer messages and the processing of those messages into data packets for transmission over the connection is performed as close as possible at the NIC that effects the transmission of the data packets over the wire. Furthermore, this mechanism can avoid implementing the entirety of the connection-oriented protocol in hardware at a NIC, which would be very complex and requires significant processing and memory resources at the NIC.

Preferably the parent network stack maintains control of the receive path of the connection.

Once the slave network stack has transmitted its allowance of data over the connection, the slave stack returns control to the parent stack and optionally requests permission to transmit additional data over the connection. Alternatively, such permission could be requested on behalf of the slave stack—for example, by an application **104** at the data processing system. It is also preferable if the parent stack can direct the slave stack to return control to the parent stack at any moment so as to allow the parent stack to perform its own transmission operations—for example, to transmit messages required to maintain the connection.

To give a particular example, data processing system **101** might support an electronic trading application **104** that is configured to make use of a set of trading algorithms arranged to execute at an FPGA **109** of NIC **102**. On identifying that a set of trades are to be made, application **104** can request that user-level network stack **105** permit packet processing engine **108** to transmit a specified amount of data over a TCP connection established by network stack **105** to an electronic exchange. The application could make such a request by means of software interface **106** described in relation to the first invention. FPGA **109** at the NIC can then operate autonomously, receiving data over network **107** from an electronic exchange and forming application layer bid and quote messages in response in accordance with its trading algorithms. Since packet processing engine **109** has permission and the necessary state to transmit over the TCP connection to the electronic exchange, the packet processing engine can process the application layer messages at the NIC and transmit the resulting data packets over the wire at very low latency.

The second invention and first invention can be utilised together with the slave network stack being configured to complete message templates generated by the parent network stack. In fact, the network protocol state passed from the parent stack to slave stack can be a set of one or more message templates generated by the parent network stack. Thus, the parent network stack can prepare a set of message templates which are passed to the slave stack for completion with application layer data and processing in accordance with the connection oriented protocol. The slave network stack can alternatively be configured to form message templates at the NIC in dependence on the state received from the parent network stack, though the slave network stack need not itself be supported at the NIC. The message templates could be stored in memory at the NIC, or at a memory of a message forming entity supported at the NIC (e.g. at a memory of an FPGA configured to from the application layer data).

In any of the embodiments of the present invention described herein it is advantageous to configure software interface **106** to provide a "dummy" transmission function that causes the appropriate CPU core of the data processing system to execute a transmit code path without actually causing any data to be transmitted over the network. This has the effect of ensuring that the instructions and state required to execute the code path are resident in the cache of the appropriate CPU core of the data processing system. This is advantageous because the time taken to execute a particular code path is generally highly dependent on whether the instructions that define the code path, and the state touched by the code path, are resident in the cache of the CPU core executing that code path. If the instructions and state are not resident in the cache, the code path typically executes much more slowly.

A dummy transmission function is provided that can be called by a software interface so as to cause the appropriate CPU core to, from its point of view, perform a transmission operation on the message template stored at the transmit buffer. In other words, the software interface causes the network protocol stack to process the message template in preparation for transmission by the NIC as though the message template were complete. However, the message template is not actually transmitted by the NIC and is preferably not passed to the NIC at all, by PIO or by DMA transfer. The missing portions of the message could be padded with values so as to form a "complete" message that can be processed by the stack.

More broadly, a software interface can be configured to provide a dummy transmit function and/or a dummy receive function configured to cause the appropriate CPU core to perform a transmission or receive operation (as appropriate) without actually transmitting or receiving any data at the data processing system. Such a software interface need not be operable to form a message template as described above in relation to FIGS. **1** to **3**. A dummy transmit function is con-

figured to cause execution of the transmit code path of the appropriate network protocol stack so as to bring the state required by the transmit code path of the network protocol stack into the cache of the processor that would perform the respective actual transmit operations. A dummy receive function is configured to cause execution of the receive code path of the appropriate network protocol stack so as to bring the state required by the receive code path of the network protocol stack into the cache of the processor that would perform the respective actual receive operations. These dummy functions have the effect of ensuring that the instructions and state required to execute the code path are resident in the cache of the appropriate CPU core of the data processing system.

Preferably the dummy transmit or receive functions are called by means of regular transmit or receive functions (such as send( ) or recv( ) TCP socket calls) carrying a predetermined flag that indicates to the software interface that the transmit/receive function is a dummy function. This ensures that calls to the dummy transmit/receive functions use the same entry point as regular transmit/receive function calls. No payload data need be provided with calls to a dummy transmit function, and no payload data need be returned in response to a dummy receive function. The software interface is preferably a socket API.

Preferably an application is configured to periodically invoke a dummy transmit and/or receive function of the software interface so as to increase the likelihood that the appropriate instructions and state are in the cache when required by genuine transmit or receive operations.

By providing dummy transmit/receive functions, the software interface allows an application to prepare the cache of a CPU and ensure that latency of the transmit/receive code paths is minimised.

The applicant hereby discloses in isolation each individual feature described herein and any combination of two or more such features, to the extent that such features or combinations are capable of being carried out based on the present specification as a whole in the light of the common general knowledge of a person skilled in the art, irrespective of whether s

uch features or combinations of features solve any problems disclosed herein, and without limitation to the scope of the claims. The applicant indicates that aspects of the present invention may consist of any such individual feature or combination of features. In view of the foregoing description it will be evident to a person skilled in the art that various modifications may be made within the scope of the invention.

We claim:

1. A method of transmitting data for use at a data processing system supporting an application, a software interface and a network protocol stack, the data processing system being coupled to a network by a network interface device, the method comprising:

the application:

forming one or more parts of a message for transmission;

causing a message to be generated by means of a first request to the software interface;

the software interface:

on receiving the first request, causing the network protocol stack to include said one or more parts of the message for transmission in the message and, in accordance with a predetermined set of protocols, form at least in part one or more protocol headers for the message for transmission;

and subsequently:

forming one or more further parts of the message for transmission;

causing the network protocol stack to include the one or more further parts of the message in the message and, on completion of the message for transmission, process the message in accordance with the predetermined set of protocols to complete the protocol headers; and

causing the network interface device to transmit the completed message over the network.

2. A method of transmitting data as claimed in claim **1**, wherein the step of forming one or more further parts of the message for transmission is performed by the application and the step of causing the network protocol stack to include the one or more further parts of the message in the message is effected by means of a second request made by the application to the software interface.

3. A method of transmitting data as claimed in claim **2**, wherein the application makes a series of second requests to the software interface, each second request being made on one or more further parts of the message for transmission becoming available to the application.

4. A method of transmitting data as claimed in claim **3**, wherein the final request of the second request type in the series completes the message for transmission causing the network protocol stack to process the message in accordance with the predetermined set of protocols to complete the protocol headers and the network protocol stack to transmit the completed message.

5. A method of transmitting data as claimed in claim **1**, wherein the network protocol stack comprises at least two parts: a first part supported at the data processing system and a second part supported at the network interface device; the first part being configured to perform the step of forming at least in part one or more protocol headers of the message for transmission, and the second part being configured to perform the step of processing the message so as to complete the protocol headers.

6. A method of transmitting data as claimed in claim **5**, wherein the step of forming the one or more further parts of the message for transmission is performed at a data processing engine of the network interface device.

7. A method of transmitting data as claimed in claim **6**, wherein the step of causing the second part of the network protocol stack to process the message to complete the protocol headers is performed in response to completing formation of the one or more further parts of the message for transmission.

8. A method of transmitting data as claimed in claim **5**, wherein the first part of the network protocol stack is a user-level protocol stack.

9. A method of transmitting data as claimed in claim **1**, wherein at least part of the message is held at the network interface device.

10. A method of transmitting data as claimed in claim **1**, wherein the first request includes memory references to the one or more parts of a message for transmission.

11. A method of transmitting data as claimed in claim **1**, further comprising, subsequent to the network protocol stack including said one or more parts of the message for transmission in the message, the network protocol stack performing segmentation of the message in accordance with a predetermined transport protocol such that the one or more parts of the message are transmitted over the network in one or more message segments by the network interface device.

12. A method of transmitting data as claimed in claim **11**, further comprising, subsequent to the network protocol stack including said one or more further parts of the message for transmission in the message, the network protocol stack caus-

ing those segments of the message that correspond to the further parts of the message for transmission to be transmitted over the network by the network interface device.

13. A method of transmitting data as claimed in claim **12**, wherein the step of the network protocol stack causing those segments of the message that correspond to the further parts of the message for transmission to be transmitted over the network completes the step of the network interface device transmitting the message over the network.

14. A method of transmitting data as claimed in claim **11**, wherein the predetermined transport protocol is the transmission control protocol.

15. A method of transmitting data as claimed in claim **11**, wherein the transport protocol is configured such that a message segment can carry a plurality of parts of the message for transmission that are not contiguous in the message for transmission.

16. A method as claimed in claim **15**, wherein a message segment can include one or more header fields that determine the logical position within the message for transmission of the parts of the message in the message segment.

17. A method as claimed in claim **16**, wherein the one or more header fields are formed in a TCP header option.

18. A method of transmitting data as claimed in claim **15**, wherein all of the one or more parts of the message are transmitted over the network in a single message segment by the network interface device.

19. A method of transmitting data as claimed in claim **1**, wherein at least part of the message is stored at the network interface device, that part of the message stored at the network interface device including the one or more parts of a message for transmission.

20. A method of transmitting data as claimed in claim **19**, wherein the step of the network protocol stack including the one or more further parts of the message in the message comprises transferring the one or more further parts of the message into the message using one or more programmed I/O instructions.

21. A method of transmitting data as claimed in claim **19**, wherein the step of the network protocol stack including the one or more further parts of the message in the message comprises:

for those further parts of the message that are below a predetermined length, transferring the one or more further parts of the message into the message using one or more programmed I/O instructions; and

for those further parts of the message that are above the predetermined length, transferring the one or more further parts of the message into the message by means of direct memory access by the network interface device.

22. A method of transmitting data as claimed in claim **1**, wherein the software interface is configured to provide a dummy transmission function by means of which an application can cause the network protocol stack to process the message in accordance with the predetermined set of protocols as if for transmission by the network interface device, the method further comprising, prior to the step of the network interface device transmitting the message over the network, the application periodically calling the dummy transmission function to cause the network protocol stack to process the message in accordance with the predetermined set of protocols without causing the network interface device to transmit the processed message over the network.

23. A method of transmitting data as claimed in claim **22**, wherein the step of calling the dummy transmission function comprises, prior to the network protocol stack processing the message in accordance with the predetermined set of protocols, the network protocol stack padding the message with data to fill the missing parts of the message for transmission.

24. A method of transmitting data as claimed in claim **1**, wherein the message is a logical expression of the message for transmission.

25. A method of transmitting data as claimed in claim **1**, wherein the software interface is an application programming interface.

26. A method of transmitting data as claimed in claim **1**, wherein at least part of the network protocol stack is at user level.

27. A data processing system coupled to a network by a network interface device and comprising:

an application configured to form one or more parts of a message for transmission;

a network protocol stack configured to process messages for transmission in accordance with a predetermined set of network protocols;

a software interface configured to, in response to a first request, cause the network protocol stack to form at least in part one or more protocol headers of a message for transmission;

wherein the application is configured to, on forming one or more parts of a message for transmission, make a first request to the software interface to cause the network protocol stack to form a message comprising at least in part one or more protocol headers of a message for transmission and the one or more parts of the message for transmission;

and subsequently:

the network protocol stack being configured to, on one or more further parts of the message for transmission being formed, include the one or more further parts of the message in the message and, on completion of the message for transmission, process the message in accordance with the predetermined set of protocols to complete the protocol headers and cause the network interface device to transmit the completed message over the network.

28. A data processing system as claimed in claim **27**, wherein the application is configured to form the one or more further parts of the message for transmission and to cause the network protocol stack to include the one or more further parts of the message in the message by means of a second request to the software interface.

29. A data processing system as claimed in claim **27**, wherein the one or more further parts of the message for transmission are formed at the network interface device at a data processing engine of the network interface, and the network protocol stack comprises at least two parts: a first part supported at the data processing system and a second part supported at the network interface device; the first part being configured to perform the step of forming at least in part one or more protocol headers of the message for transmission, and the second part being configured to perform the step of processing the message so as to complete the protocol headers.

30. A method of transmitting data for use at a data processing system and network interface device, the data processing system being coupled to a network by the network interface device, the method comprising:

forming a message in accordance with a predetermined set of network protocols, the message including at least in part one or more protocol headers comprising header data; and subsequently:

developing an application layer message in one or more parts;

updating the formed message with the parts of the application layer message;

processing the formed message in accordance with the predetermined set of network protocols to complete the protocol headers; and

causing the network interface device to transmit the completed message over the network.

31. A method of transmitting data as claimed in claim **30**, wherein the step of forming a message is performed at a network protocol stack supported at the data processing system.

32. A method of transmitting data as claimed in claim **30**, wherein the step of forming an application layer message is performed by one of an application supported at the data processing system, and a data processing engine of the network interface device or other peripheral device of the data processing system.

33. A method of transmitting data as claimed in claim **30**, wherein the processing step is performed at a packet processing engine supported at the network interface device.

34. A data processing system coupled to a network, the data processing system comprising:

a network protocol stack configured to, in accordance with a predetermined set of network protocols, form a message including at least in part one or more protocol headers comprising header data; and

a message forming engine configured to develop an application layer message in one or more parts and, subsequently to the network protocol stack forming a message including at least one protocol header comprising header data, cause the one or more parts of the application layer message to be applied to the formed message;

wherein the network protocol stack is further configured to, on completion of applying the one or more parts of the application layer message to the formed message, process the formed message in accordance with the predetermined set of network protocols so as to complete the protocol headers and cause the completed message to be transmitted over the network.

35. A data processing system as claimed in claim **34**, wherein the network protocol stack is supported at a network interface device arranged to couple the data processing system to the network.

36. A data processing system as claimed in claim **34**, wherein the message forming engine is one or both of an application supported at the data processing system or a data processing engine at a network interface device arranged to couple the data processing system to the network.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 9,258,390 B2
APPLICATION NO. : 13/283420
DATED : February 9, 2016
INVENTOR(S) : Steven L. Pope et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

In the Claims

at column 18, claim 5, line 35, delete "so as";

at column 20, claim 29, line 57, delete "so as".

Signed and Sealed this
Third Day of May, 2016

Michelle K. Lee

Michelle K. Lee
Director of the United States Patent and Trademark Office